

How to extract data from proprietary software database systems using TCP/IP?*

Marc Burdon[†](burdon@cs.uni-bonn.de).[‡]

December 2011 (publishing April 2014)[§]

1 Abstract

This document is a **white paper**¹ about how to connect reverse engineering and programing skills to extract data from a proprietary implementation of a database system to build EML-Tools[1] for data format conversion into raw data.

This article shows how to access data of a source software system without any interface for data conversion. We discuss how raw data can be transfered into structural format by using XML or any other custom designed software solution. For demonstration purposes only, we will use a CRM[2] system called *Harmony*[®] by Harmony[®] Software AG², the programing language *Python* and methods of computer security, which are used to get quick access to the raw data.

2 Requirements

- Python, programing skills in Python³ and in network programing[3];
- Reverse Engineering skills[4];
- XML knowledge (optionally recommended)⁴[5];
- Wireshark⁵ and one trail copy of Harmony[®].

*This copyright protected article is for scientific purpose only. According to German law this scientific document is not supposed to act or be abused against §202 StGB.

[†]Dipl.-Inform. Marc Burdon is a computer scientist, who graduated at the University of Bonn (Germany).

[‡]Copyright ©2014 by Marc Burdon.

[§]This paper was written in December 2011, but refreshed and published now, in April 2014.

¹This paper (Version May 13, 2014) does not give any scientific progress, but it shows potentials of the chosen topics on higher scientific level.

²All trademarks are property of their owners, as Harmony[®] is of Harmony Software AG.

³Please refer to docs.python.org

⁴Please refer to www.w3.org

⁵Please refer to www.wireshark.org

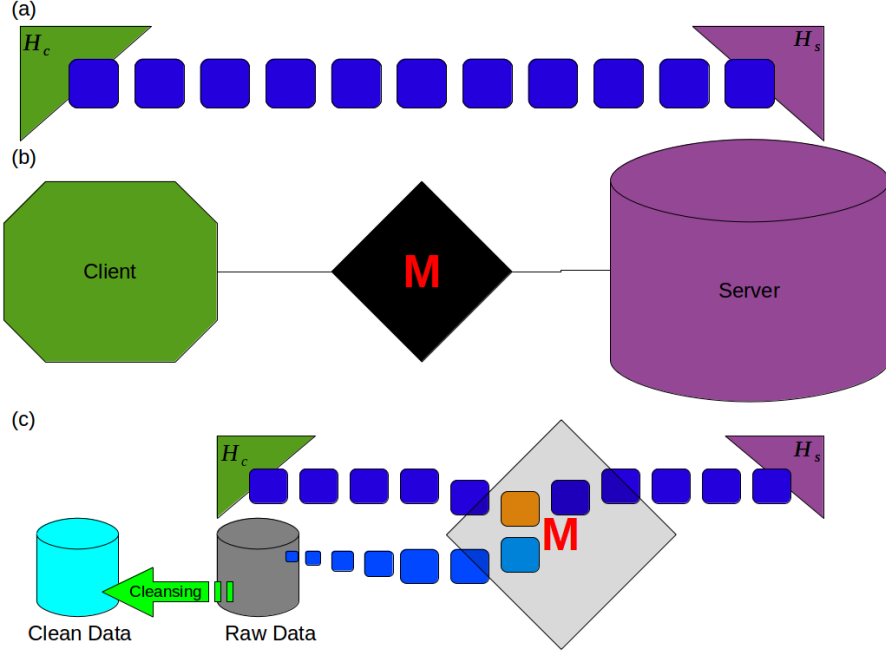


Figure 1: Figure (a) shows the regular data package streams between client and server. During Man-in-the-middle attack (b) data packages will be copied and stored as raw data (c). When data extraction is finished, the data is ready to be migrated to another system e.g. a SQL database.

3 Theory

3.1 Method to Access Data

In the beginning, we realize that offline data extraction from files on the file system of the database server will cost much more time than we will need using *TCP/IP* technology, because the reverse engineering process is much more quicker in this case, which is using the *TCP/IP* client of Harmony[®] CRM software solution. We have to know the structure of the Harmony[®] CRM software system that we just name H . The relevant core of H consists of a *TCP/IP* client and a *TCP/IP* server, which is the database server. We name Harmony[®] Software client as H_c and Harmony[®] software database server as H_s . We define a data package $d = (b_0, \dots, b_n)$ with b_0, \dots, b_n , which are bytes coding a string character. We also define the set of data packages D and a set of raw data R . We also use a self made network package filter $M(\cdot, \cdot)$. $M(\cdot, \cdot)$ is actually a function mapping data packages transferred by using *TCP/IP* from H_c and H_s (and vice versa) to R , so formally $M, M : D \times D \rightarrow R$, is commutative. We use M for a *Man-in-the-middle* attack[6] on H . Technically M is a hybrid

TCP/IP of both client and server system, which is H_c is connected to and H_s is connected by. The data connection between H_c and H_s is not encrypted by default. It can be encrypted by using e.g. *TLS/SSL* technology – though the functionality is implemented by using *SSH*[7][8] –, but by default in local area networks traffic is not supposed to be encrypted by manufacturer.

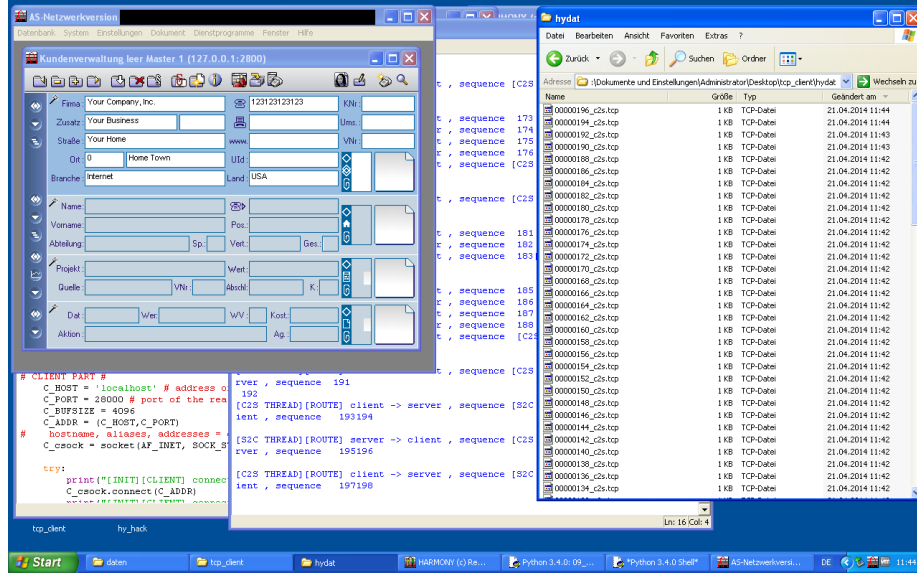


Figure 2: This is a screenshot of the Desktop of the machine that is hosting H and M (09_tcp_hyrouter.py). At his point the connections between H_c and M as well as M and H_s are established. M is capturing all TCP/IP data packages, which are sent from H_c to H_s .

SSH tunneling is activated for remote data exchange only. In fact, because of missing encryption it is easy to interpose M and run a Man-in-the-middle attack.

The client server protocol of H also misses handshake based authentication procedure that uses cryptographic functions. It is only string based, meaning to make M establish a connection to the server we just easily have to sniff – e.g. using wireshark – the first data packages to get the string, which consists of user name and password. The string can establish communication to the database server without even logging into the system. The system looks up the user in its user list and verifies the password to grant access.

3.2 Extraction of Raw Data

The reverse engineering process is easy since M is implemented and runs interpolated between H_c and H_s . Once we are connected to the server via M , M can log network transmission traffic, so client commands, which gets sent to

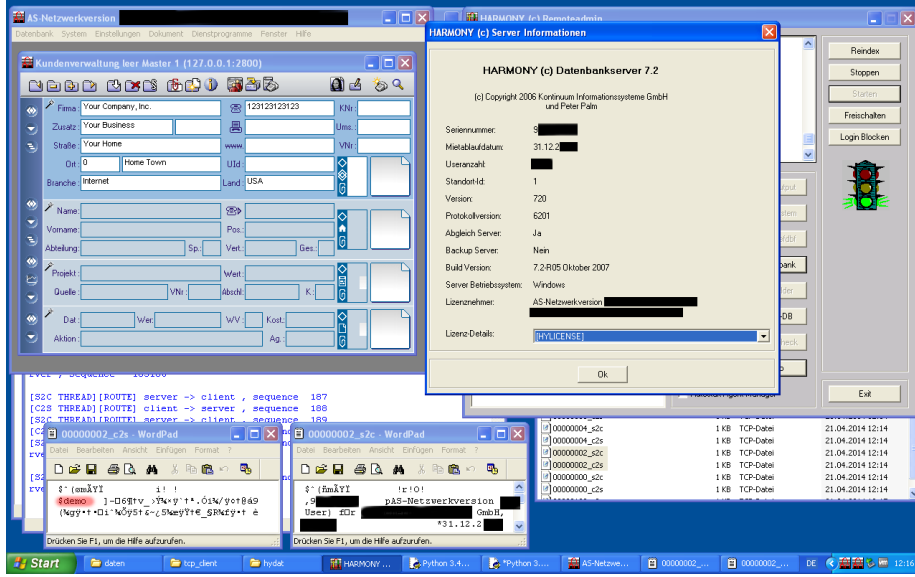


Figure 3: Each transferred TCP/IP package has been captured by M and saved to the hard disk (see right corner). The user name *demo* and the licence information have been transferred in plain text and can be captured.

the server, can be identified. This process is very easy to handle, since on each pressed button of the GUI of H_c the client sends the command containing data packages over the established TCP/IP connection.

Obviously M can also be used as standalone client without being connected by client, meaning formally $M(\emptyset, D')$ with a generator function, which generates fake packages from captured data, or $M(D_f, D')$ with D_f is the set of fake data packages, which contains server and database control commands, and D' consisting of data packages from H_s . To get a proper D_F we have to reverse engineer the full native protocol of H .

This can be done by using H_c on M to connect to H_s as described above.

To keep work time as low as possible, we skip reverse engineering of the full proprietary protocol of H . Instead, we just take the most important client commands that transfer data to the client to display it to the user. The traffic contains the raw data, we are looking for. M will copy and save the raw data on-the-fly.

3.3 Data Cleansing

Finally, the collected raw data R is full of non-data relevant symbols, so *Data Cleansing* (dc) is required for proper data extraction. Therefore we have to identify the introduction and the termination bytes of the data. We have to define a structure that will be build by the introduction and termination bytes, which R

Simple Algorithm for Data Cleansing

Input: Set C of m known data bytes $C = \{c_0, \dots, c_{m-1}\}$
Set of n raw data bytes $R = \{r_0, \dots, r_{n-1}\}$
Output: Clean and structured data in bytes as a set S_{XML}

init empty set S_{XML} ;
do $S_{XML} = \text{build}_{\text{structure}}(R, C)$
return: S_{XML}

Figure 4: Assuming that we know the bytes, which introduce and terminate data, we manage them in a set called C with $C \subset R$. We can use this algorithm for data cleansing. Function $\text{build}_{\text{structure}}(\cdot, \cdot)$ constructs the data structure by setup filtering C and returning $R \setminus C$ in XML syntax as S_{XML} .

contains. For our algorithm (see Figure 4) we define a function $\text{build}_{\text{structure}}(\cdot, \cdot)$ that builds the structure of the extracted data and removes given introduction and termination bytes. The structure of the extracted data is usually based on tree-like structures and/or SQL table schemes, when migrating data to SQL database. On the other hand it is also possible to extract the data into the file system, meaning to store the text data in text files and binary data like pictures as binary image files. So the design of the structure depends on the application. The implementation of M , which is based on the prototype *09_tcp_hyrouter.py* will not extract data. *09_tcp_hyrouter.py* is supposed to capture network traffic only to support reverse engineering process of the transfer protocol of H in this paper.

Corollary 3.1. *Simple Algorithm for Data Cleansing needs $O(n^2)$ steps to terminate.*

Proof. We have two sets R and C with $|C| < |R|$, because of $C \subset R$, so initiation is

$$|C| \cdot |R| = m \cdot n = O(n^2).$$

Obviously structural analysis takes $O(n^2)$, because we have max. of $m \cdot n$ steps. Further more, constructing XML based tree structure from data takes $O(n^2)$, because the function has to check input and refer to predefined actions regarding to introduction and termination bytes, so the procedure is powered by n in the length of the input, meaning $O(n^2)$. Simple Algorithm for Data Cleansing is terminating, because C and R are not infinite. It runs correct, because all byte coded string symbols are predefined and known to the algorithm. Finally, we get

$$3 \cdot O(n^2) = O(n^2)$$

by definition. □

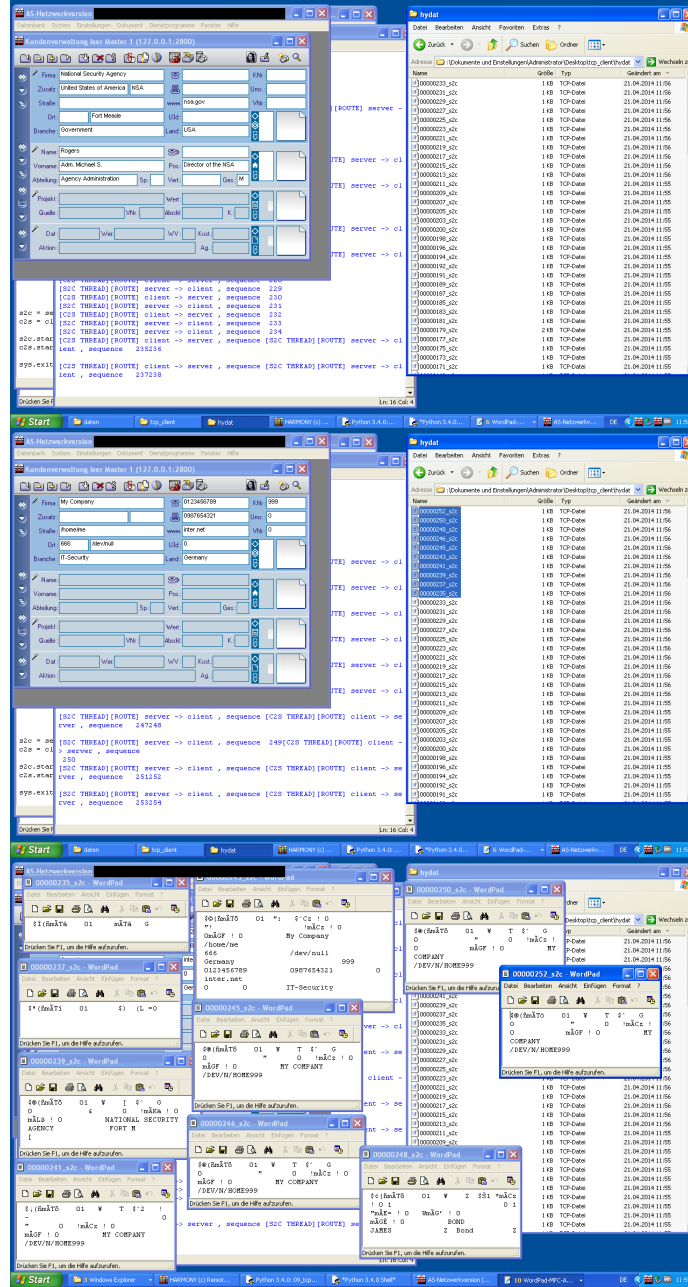


Figure 5: Each transferred TCP/IP package has been captured and stored to local the hard disk drive. Data was transferred in plain text.

4 Practice and Future Work

A demo version of *M* a.k.a. 09_tcp_hyrouter.py written in C/C++ is about TBA for scientific purposes.⁶ A detailed description of the transfer protocol of *H* is available in the full report of *How to extract data from proprietary software database systems using TCP/IP*. The full reverse engineered description of the transfer protocol of *H* is TBA. Regarding demonstrative details, please refer to Figure, 2, 3 and 5. For detailed information about this topic, please refer to the full report of *How to extract data from proprietary software database systems using TCP/IP*.⁷

M will be redesigned and implemented in C/C++ with additional features like data cleansing function and compatibility to XML. It will automatically convert the output in XML, so data migration to SQL will be possible. Additionally there will be a data extraction mode, which will optionally store the raw or fully extracted data on the hard disk drive – meaning a data cleansing procedure will be executed for full data extraction. The implementation will be available for Windows®, Linux®, MacOSX® and NetBSD.

References

- [1] H. K. Arndt and O. Günther. (2000) Environmental Markup Language (EML): Workshop 1. Metropolis.
- [2] W. Schwetz. (2001) Customer Relationship Management. Mit dem richtigen CRM-System Kundenbeziehungen erfolgreich gestalten. Gabler Verlag.
- [3] D. E. Comer. (2013) Internetworking with TCP/IP, Volume One - 6th Edition. Prentice Hall, Inc.
- [4] E. Eilam. (2005) Reversing: Secrets of Reverse Engineering. Wiley Publishing, Inc.
- [5] E. R. Harold and W. S. Means. (2004) XML in a Nutshell, 3rd edition. O'Reilly.
- [6] R. Schifreen. (2006) Defeating the Hacker: A non-technical Guide to Computer Security. John Wiley and Sons.
- [7] Internet Engineering Task Force. Network Working Group. [Online]. Available: <http://tools.ietf.org/html/rfc4253>
- [8] Internet Engineering Task Force. Network Working Group. [Online]. Available: <http://tools.ietf.org/html/rfc6668>

⁶Please refer to cs.burdon.de/downloads for further information.

⁷Please refer to cs.burdon.de/hy